Embedding EUROPA

EUROPA comes with a tool called makeproject, which will generate C++ and Java projects for you.

They both illustrate how you can perform the full application cycle :

1. Initialize EUROPA
2. Load/Modify model and initial state descriptions
3. Invoke a solver
4. Extract plan results from the Plan Database
5. Repeat steps 2-4 as many times as needed
6. Shutdown EUROPA

Using the C++ API

Take a look at the main() program that makeproject generates for the C++ project : Main.cc

You have 2 options to implement the application cycle described above :

- The PSEngine interface is the official interface for EUROPA clients, it is the recommended way to use EUROPA. This interface is very straightforward and allows you to run the entire application cycle described above. This abstraction layer will isolate your client code from most changes in the internals of the EUROPA implementation, it is also designed for easy mapping to other languages using SWIG, if you're planning to write your application in a language other than C++ this interface should be either already available in the EUROPA distribution, or relatively easy to add (currently only Java bindings are bundled with the EUROPA distribution, but we have plans to add Python and any other languages that are popular with the EUROPA user base).
- The EuropaEngine interface gives access to the internal modules of EUROPA. You will have to spend more time understanding the different classes, probably write more code to extract information from the Plan Database and be more careful about the calls that you make. EuropaEngine is a base class to the PSEngine instances that you get from PSEngine::makeInstance() calls, so you can always dynamic_cast a PSEngine instance to a EuropaEngine one. By using EuropaEngine you will not be isolated from changes in the EUROPA internals, therefore you should be ensure that your needs are not met by the PSEngine API before using this interface.

You will probably want to start with the PSEngine interface and if it is doesn't give you sufficient low-level access (this should be rare, except for very advanced applications) for your purpose switch to the EuropaEngine interface (just do a dynamic cast as described above).

Eventually the PSEngine interface will be extended to expose all the extension points in EUROPA and external clients should never have to use EuropaEngine.

Using the JAVA API

The PSEngine interface is automatically mapped to Java using SWIG
Take a look at the main() program that makeproject generates for the Java project : Main.java
makeproject uses a combination of Java and BeanShell scripting to achieve its goal, if you put everything together in a single Java program it would look something like this :

```
import org.ops.ui.util.LibraryLoader;
import psengine.*;

class Main
{
    public static void main(String args[])
    {
        String debugMode = args[0];
        String nddlFilename = args[1];

        PSEngine europa = makePSEngine(debugMode);
        europa.start();
        europa.executeScript("nddl",nddlFilename,true/*isFile*/);
        runSolver(europa);
        europa.shutdown();
   }

    /*
     * debugMode = "g" for debug, "o" for optimized
     */
    static PSEngine makePSEngine(String debugMode)
    {
        PSEngine psEngine;
        LibraryLoader.loadLibrary("System_"+debugMode);
        psEngine = PSEngine.makeInstance();

        return psEngine;
    }

    static void runSolver(PSEngine europa)
    {
        String plannerConfig = "PlannerConfig.xml";
        int startHorizon=0, endHorizon=100;

        PSSolver solver = europa.createSolver(plannerConfig);
        solver.configure(startHorizon,endHorizon);

        int maxSteps = 1000;
        for (int i = 0; !solver.isExhausted() && !solver.isTimedOut() && i<maxSteps; i = solver->getSte
            solver.step();
            if (solver.getFlaws().size() == 0)
                break; // we're done!
        }

        if (solver->isExhausted())    debugMsg("Solver was exhausted after " + i + " steps");
        else if (solver.isTimedOut()) debugMsg("Solver timed out after " + i + " steps");
        else                          debugMsg("Main","Solver finished after " << i << " steps");
    }
```

```
    static void debugMsg(String msg)
    {
        System.out.println(msg);
    }
}
```

Building your own project

If you don't want to use the infrastructure generated by makeproject, you will need to :

- C++ API
  - Add the directories $EUROPA_HOME/include and $EUROPA_HOME/include/PLASMA to your include path
  - Link in the EUROPA libraries from $EUROPA_HOME/lib.
- Java API
  - Add the following files from $EUROPA_HOME/lib to your classpath : nddl.jar, PSEngine.jar, PSUI.jar

In both cases, make sure $EUROPA_HOME/lib is in your LD_LIBRARY_PATH (or on your PATH if you're on Windows)